Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 96 (2016) 961 – 967
Procedia Computer Science

20th International Conference on Knowledge Based and Intelligent Information and Engineering Systems

A Dynamic Editor of Typed Data Transformations

Ruban S.I.[a], Dr. Shapkin P.A.[a,b,*], Dr. Zykov S.V.[b,a]

[a] *National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Department of Cybernetics and Information Security, Kashirskoe shosse, 31, Moscow and 115409, Russia*
[b] *Innopolis University, 1 Universitetskaya str. Innopolis, 420500, Russia*

Abstract

This paper presents tools aimed at automating the data integration process between different SaaS-applications. Data transformations creation and customization represents a major task in this area. We propose a solution based on type theory and describe an approach to automated transformation construction in form of typed functions.

Peer-review under responsibility of KES International
Keywords: Type theory, Cartesian product, transformation rules

1. Introduction

The main problem of the data integration process is the construction of data transformations. We propose an approach to creating a transformation editor based on type theory. We derive the scenarios of constructing transformations from the corresponding typing rules. The obtained system is able to filter properties unusable within the current type of transformation. This feature improve usability and more over decrease probability of creating incorrect transformations between incompatible properties.

The article is organized as follows. In the second section the structure of transformations is formalized. In the third section we analyse typing rules in order to propose the scenarios of user interaction during the transformation construction. Section four discusses some implementation details and the conclusion summarizes the obtained results.

2. Type-Theoretic Structure of Data Objects

We introduce the several notions which be used later.

Notion. A *type system* is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute. Thus the type system for-

* Corresponding author. Tel.: +7 (903) 731-42-72.
E-mail address: pashapkin@mephi.ru

Peer-review under responsibility of KES International
doi:10.1016/j.procs.2016.08.094

mally offers type inference and checking research. The programs are presented as composition of computable functions. We introduce a notion of type.

Notion.

- If V is a type variable or constant, then V is type
- If V and U is type, then $V \to U$ is type

The cast data type needs appear during processing program. But it is impossible to construct object transformation in common case. We assume that each domain entity has appropriate type. We consider the notion of type as records with a set of named fields in the present context. The fields be understanding as getters (the function which extract values from object) [1]

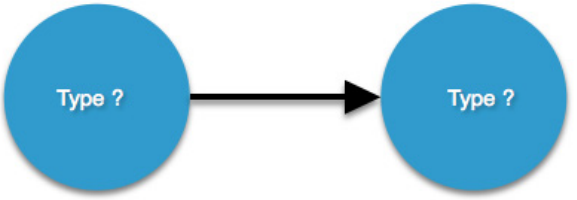

Figure 1. General case of type transformation

It is impossible to construct transformation function in case of unknown types if the function didn't specified initially. Therefore the particular data type cases are considered.

Before that a range of new notions is introduced.

Notion.

Property is a complex the following field: data type *tpe*, *extractor* and *wrapper* functions.

Extractor is a function that takes as input an object of *tpe* type and extracts from it a property of certain type, which can cause the input property.

Wrapper is a function that takes a property of certain type, which can lead to an object of *tpe* type, and wraps it in an object of *tpe* type.

We explain a specific example.

We assume that P_1 property is a property with a value of type T. It's mean that e_{P_1} is a function that takes as input an object of P_1 type and returns an object of T type, i.e

$$e_{P_1} : P_1 \to T$$

and w_{P_1} is a function that takes as input an object of T type and returns an object of P_1 type, i.e

$$w_{P_1} : T \to P_1$$

The fact that P is a property with type value T is written as $P : \mathbb{P}[T]$.

We assume that it is possible to construct one-sided transformation function which convert property P_1 to property P_2. It's mean that a function t_{12} that take as input an object of P_1 type and returns an object of P_2 type exists, i.e $t_{12} : P_1 \to P_2$ However, the existence of the transformation function from P_1 property to P_2 property does not ensure that the existence of transformation function from P_2 property to P_1 property. For example, a such transformation function exists between the properties of "date of birth" and "age" of, but not vice versa.

Notion.

The *properties composition* is a data type which is a cortege of properties.

We now consider the the particular cases of transformation functions construction.

1. If input and output properties have the same data type then data transformation can be carried. A transformation rule for this case will be as follows:

$$\frac{P_1 : \mathbb{P}[T] \quad P_2 : \mathbb{P}[T]}{w_{p2} \circ e_{p1} : \quad P_1 \to P_2} \quad (1)$$

where

$P : \mathbb{P}[T] \Longleftrightarrow (\exists e_p(e_p : P \to T)) \wedge (\exists w_p(w_p : T \to P))$

2. If input and output properties don't have the same data type but transformation function for corresponding data types was determined initially then data transformation can be carried. The transformation function for this case will be as follows:

$$w_1 * e_2 : P_2 \to P_1$$

A transformation rule for this case will be as follows:

$$\frac{P_1 : \mathbb{P}[T_1] \quad P_2 : \mathbb{P}[T_2] \quad t : T_2 \to T_1}{w_{p2} \circ t \circ e_{p1} : \quad P_1 \to P_2} \quad (2)$$

3. If the objects are properties composition then transformation is possible if and only if the transformation functions known to all types of components.

This case is divided into two others:

- The number of properties in the input object equals the number of properties in the output object
- The number of properties in the input object doesn't equal the number of properties in the output object

Consider the example of the first case: Input type:

$$T_1 : A_1, A_2, ..., A_n$$

Output type:

$$T_2 : B_1, B_2, ..., B_n$$

The transformation functions for corresponding types:

$$t_1 : A_1 \to B_1$$
$$t_2 : A_2 \to B_2$$
$$...$$
$$t_n : A_n \to B_n$$

The transformation function for this case will be as follows:

$$t_1 \times t_2 \times ... \times t_n : A_1 \times A_2 \times ... \times A_n \to B_1 \times B_2 \times ... \times B_n$$

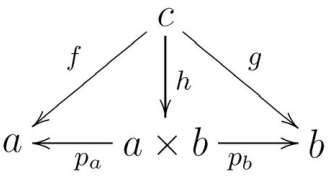

Figure 2. Product

Thus it is possible to consider the properties composition as a particular case of the Cartesian product:
Notion.
The *product of object* a and b is an object $a \times b$ with morphisms $p_a : a \times b \to a$ and $p_b : a \times b \to b$ there exists a unique morphism $h : c \to a \times b$ for any object c with morphisms $f : c \to a$ and $g : c \to b$that the diagram is commutative. The morphism $p_a : a \times b \to a \quad p_b : a \times b \to b$ is a *projection*.

However, in the case of properties composition the order of the elements does not matter: for Obj object that has the properties of P_1 and the P_2, is identically equal to the expressions:

$$P_1 \times P_2 \to Obj \quad P_2 \times P_1 \to Obj.$$

Therefore we introduce new operation is $*$ which by definition, is identically equal to the Cartesian product without taking into account the order of the arguments. The second case also requires a transformation function for all component of the type that automatically reduces to the existence of functions of transformations:

$$t : A_{x_1} \times ... \times A_{x_m} \to B_z$$
or
$$t : A_x \to B_{y_1} \times ... \times B_{y_m}$$

Notion
We will call a function as *trivial*, if the transformation of the same type is occurred. For example, if a property name1, which has type String, is casted to a property name2, which also has type String, then such transformation is trivial. Otherwise we will call a function as *nontrivial*.[2]

3. Formalizing Transformation Construction Scenarios

There is the more serious problem when the next case is appeared. There is several object obj_1 which has property (field) $pr_1 : T_1$ and it is necessary to transfer data from pr_1 to $appl_2$ for the further processing. But there aren't objects with properties which has type T_1 in $appl_2$. For example, in $appl_1$ the property phNum_1 is contained the phone number as number: 87584509643. There aren't objects with properties which has type T_1 in $appl_2$, but there is a property $phNum_2$ which has type String and satisfy several regular expression. Due to this expression the data property is looked as such view: $X(XXX)XXX-XX-XX$. Obviously, if $phNum_1$ contains integer positive number which is started at "8", then it is possible to create the several transformation function. This function record property $phNum_2$ in the correct format. To solve a question, we develop the schema of transformation.

The considerable contribution of successful application is the correctly constructed interface. The interface is the one available things to user. The UI development is proceed the same way to make it more attractive and comfortable for optimizing the user interaction. Thus we must pay due attention UI construction during application development and analyse designed pattern of application view, test usability and refactor and error correction based on the received data. The pattern of application was designed as a result to analyse data which were received from user. We analyzed the input data for each of the the particular cases the transformation of the above.

1. According to the rule(1), the user must have the following data were obtained:

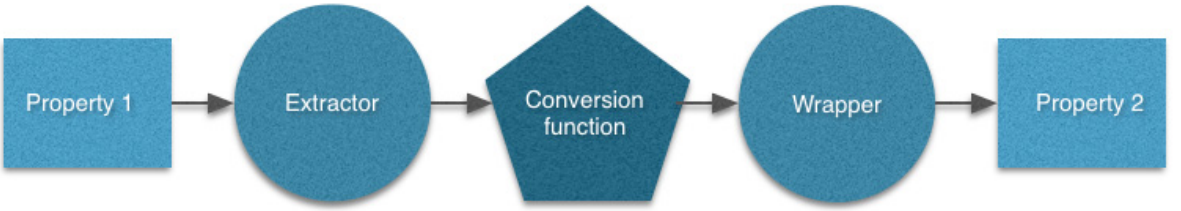

Figure 3. Schema of transformation

- Input property
- Output property

The order of data entry does not affect the final result, but when any of the properties narrows the range of possible values of the second property.

2. According to the rule (2), the user, as in the first case, enters the two properties; however the system stores previously defined transformation function between the properties of of the corresponding types. In this case there are 6 variants of the sequence selecting transformation elements:

   - $P_1 \triangleright t \triangleright P_2$.When the first property is choosing the range of possible functions of transformation t narrows(there are only those functions that take input from the properties of the same type as that of P_1) and the range of candidates for the second property narrows (there are only those properties whose types are included in the number of output properties remaining transformation functions). After selecting t there is only such properties whose type is the type of the output properties of t.
   - $P_1 \triangleright P_2 \triangleright t$. The case of the selection of the first properties described above. After selecting the second property there is a unique variant of choice function transformations.
   - $P_2 \triangleright t \triangleright P_1$. The case is similar to the first, however, the transformation functions with the output type that matches the type of the P_2 remain, and candidates for the role of the first properties are properties with a type equal to the input of the selected type transformation functions t.
   - $P_2 \triangleright P_1 \triangleright t$. This case is similar to the second case to within change the input / output property.
   - $t \triangleright P_1 \triangleright P_2$.Selection of transformation function immediately cuts selection options of input and output properties according to the input and output types t, after selecting a candidate list P_1 to be the second property is not changed.
   - $t \triangleright P_2 \triangleright P_1$. The case is similar to the previous case.

3. This case can be reduced to the first two, but it is complicated by the choice of multiple features and transformations functions for each property in the object.

The name of application is specified on the top part of page. The properties are separated into two columns, depending on the role performed: input properties locate on the left part of the page, output properties locate on the right part of the page. The same placement of elements is caused by the existing model of reading from left to right among the future users of the application. For an unambiguous choice properties as controls were selected radio boxes that exclude ability to select multiple items in a single subgroup. The two groups were separated:

1. Input properties

2. Output properties

The button "*Processed*" is located on the bottom of page. The its task is the launching transformation mechanism for selected properties. The warning as the pop-up window will be printed if a user pressed this button but didn't select two properties. The design pattern has simple and intuitive user view, but the button "?" was added. When the user press on it the hint with user instruction is appeared.

4. Implementation Details

The platform Tylip expected to unite of the different SaaS services and escalate data migration between include services. Thus it is proposed that access to the Internet is be provided for work with platform. But we must considered the fact that the building transformation functions can be need to user which doesn't have access to the Internet. In this case the ability to run a standalone application must be provided. The libraries which is loaded in progress application start is the one developer's problem. To resolve this problem we come to a conclusion to use solution WebJars.

WebJars is a set of libraries. Each library is a JavaScript library or a CSS module. WebJars provides a simple and understandable way to control dependencies in web application, is compatible with the project build tools implemented for VM-based language such as SBT, Maven and etc. All libraries is loaded from central Maven repository.

So, this approach allows to avoid another problem which wasn't mentioned previously. It is arose on condition specifying a reference to an external hosting of libraries and long period of the software maintenance. It is a loss of library as the result of external site problems. All need dependencies are loaded In case of using WebJars. WebJars provides an easy control version of include libraries as an additional capability. All current versions is available in WebJars interface. Play Framework was selected as the base framework. It provides equipment developing and quick deploying of web-applications, allows programming on Scala language, which was selected to implement unified platform tylip, supports many javascript frameworks, such as AngularJS, CoffeScript, ReactJS etc. Finally JS framework AngularJS was chosen, which intending to one-page web-application developing.

The platform tylip structure should be presented before the description of Transformation editor application integration schema. Project consist of the follow modules:

- adapters - sources of connectors to different connecting systems (Bitrix24, My store).
- tylip-core - the tulip platform core. Consist of: terms, transformation functions etc., which was implemented to converse data types.
- tylip-web - web-application module, its structure is the same to structure of application to PlayFramework.
- project tulip-platform-build - contain project build setting files, files with system variables values, files with application launch parameters.

So, transformation editor integration in tylip reduces to corresponding classes implementation into tylip-web module. The html files (*index.scala.html* and *main.scala.html*) are moved to *app/views/Properties* subdirectories. The controllers (*controllers.js* and *propertiesFactory.js*) are moved to *public/javascripts/properties* directories. The styles file (property.css) are moved to *public/stylesheet/properties*.

The class *Application.scala* is a Controller trait and responses to transformations business-logic in transformations editor. It places in *app/controller* path. The class, which is property model and contains extractors and wrappers, locates in *app/models/properties* directory.

The same way the used pictures, styles and other files are embedded in application binded an external view.

The routes file responses to routing and navigation in application. There must be only one instance in application, therefore it is necessary to move strings response to new routed paths into correspond files

5. Conclusion

The scheme of type safe transformations was found. User functionality was implemented, according to completed investigations on optimal elements placement field. Further we are planning to complete transformation editor implementation for non-trivial transformation functions and supply according user functional, to improve user tips system, to develop user functions storage scheme based on the chosen data base.

6. Acknowledgement

The research was supported by a Russian Foundation for Basic Research grant (Project № 16-31-50011) and the Competitiveness Program of NRNU “MEPhI”.

References

1. P. Shapkin, G. Pomadchin. A Type-Theoretic Approach to Cloud Data Integration // Proceedings of the 11th International Conference on Web Information Systems and Technologies WEBIST. —INSTICC Press, 2015. —May. —P. 164–169.
2. Pierce B. C. Types and programming languages. —The MIT Press, 2002.